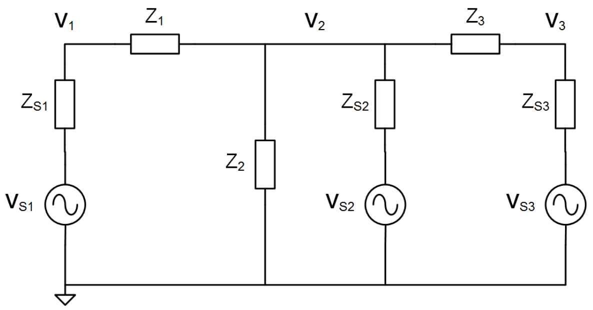
SECTION 5: POWER FLOW

ESE 470 – Energy Distribution Systems

Introduction

Nodal Analysis

- Consider the following circuit



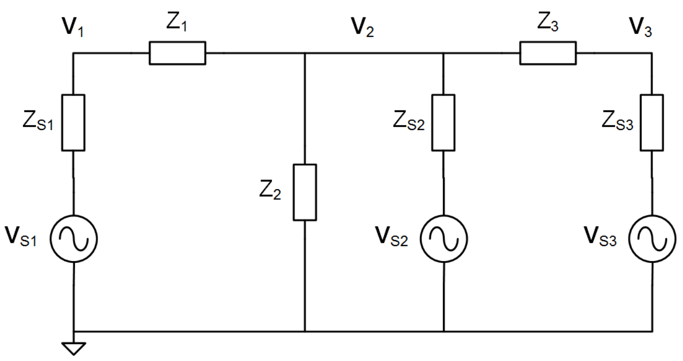
- Three voltage sources
  - V_{s1}, V_{s2}, V_{s3}
- Generic ***branch impedances***
  - Could be any combination of R, L, and C
- Three unknown ***node voltages***
  - V_1, V_2, and V_3
- Would like to analyze the circuit
  - Determine unknown node voltages
- One possible analysis technique is ***nodal analysis***

Nodal Analysis

- **Nodal analysis**
  - Systematic application of ***KCL*** at each unknown node
  - Apply Ohm's law to express branch currents in terms of node voltages
  - Sum currents at each unknown node

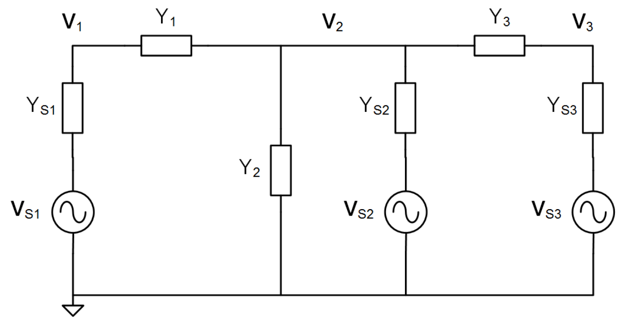
- We'll sum currents *leaving* each node and set equal to zero
- At node V_1, we have

$$\frac{V_1 - V_{s1}}{Z_{s1}} + \frac{V_1 - V_2}{Z_1} = 0$$



- Every current term includes division by an impedance
  - Easier to work with ***admittances*** instead

Nodal Analysis



- Now our first nodal equation becomes

$$(V_1 - V_{s1})Y_{s1} + (V_1 - V_2)Y_1 = 0$$

where

$$Y_{s1} = 1/Z_{s1} \quad \text{and} \quad Y_1 = 1/Z_1$$

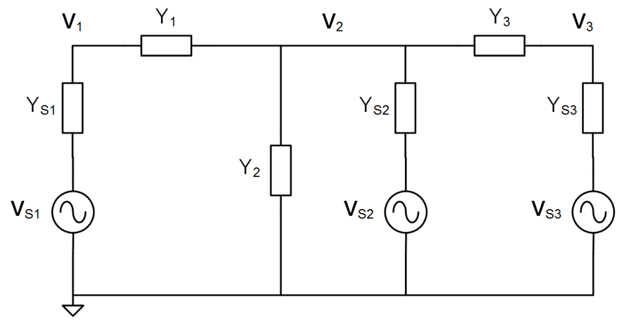
- Rearranging to place all unknown node voltages on the left and all source terms on the right

$$(Y_{s1} + Y_1)V_1 - Y_1 V_2 = Y_{s1} V_{s1}$$

- Applying KCL at node V_2

$$(V_2 - V_1)Y_1 + V_2 Y_2 + (V_2 - V_{s2})Y_{s2} + (V_2 - V_3)Y_3 = 0$$

Nodal Analysis



- Rearranging

$$-Y_1V_1 + (Y_1 + Y_2 + Y_{s2} + Y_3)V_2 - Y_3V_3 = Y_{s2}V_{s2}$$

- Finally, applying KCL at node V_3, gives

$$(V_3 - V_2)Y_3 + (V_3 - V_{s3})Y_{s3} = 0$$

$$-Y_3V_2 + (Y_3 + Y_{s3})V_3 = Y_{s3}V_{s3}$$

- Note that the source terms are the ***Norton equivalent current sources*** (***short-circuit currents***) associated with each voltage source

Nodal Analysis

- Putting the nodal equations into matrix form

$$\begin{bmatrix} (Y_{s1}+Y1) & -Y_1 & 0 \\ -Y_1 & (Y_1+Y_2+Y_{s2}+Y_3) & -Y_3 \\ 0 & -Y_3 & (Y_3+Y_{s3}) \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} Y_{s1}V_{s1} \\ Y_{s2}V_{s2} \\ Y_{s3}V_{s3} \end{bmatrix}$$

  or

$$\boldsymbol{YV} = \boldsymbol{I}$$

  where
  - $\boldsymbol{Y}$ is the $N \times N$ admittance matrix
  - $\boldsymbol{I}$ is an $N \times 1$ vector of known source currents
  - $\boldsymbol{V}$ is an $N \times 1$ vector of unknown node voltages

- This is a system of N (here, three) ***linear equations*** with N unknowns
- We can solve for the vector of unknown voltages as

$$\boldsymbol{V} = \boldsymbol{Y}^{-1}\boldsymbol{I}$$

The Admittance Matrix, $\boldsymbol{Y}$

- Take a closer look at the form of the admittance matrix, $\boldsymbol{Y}$

$$\begin{bmatrix} (Y_{s1}+Y1) & -Y_1 & 0 \\ -Y_1 & (Y_1+Y_2+Y_{s2}+Y_3) & -Y_3 \\ 0 & -Y_3 & (Y_3+Y_{s3}) \end{bmatrix} = \begin{bmatrix} Y_{11} & Y_{12} & Y_{13} \\ Y_{21} & Y_{22} & y_{23} \\ Y_{31} & Y_{32} & Y_{33} \end{bmatrix}$$

- The elements of $\boldsymbol{Y}$ are
  - Diagonal elements, Y_{kk}:
    - $Y_{kk} =$ sum of all admittances connected to node k
    - ***Self admittance*** or ***driving-point admittance***
  - Off-diagonal elements, Y_{kn} $(k \neq n)$:
    - $Y_{kn} = -$(total admittance between nodes k and n)
    - ***Mutual admittance*** or ***transfer admittance***
- Note that, because the network is reciprocal, $\boldsymbol{Y}$ is symmetric

Nodal Analysis

- Nodal analysis allows us to solve for unknown voltages given circuit admittances and current (Norton equivalent) inputs
  - An application of ***Ohm's law***

$$\boldsymbol{YV = I}$$

  - A ***linear equation***
  - Simple, algebraic solution
- For power-flow analysis, things get a bit more complicated

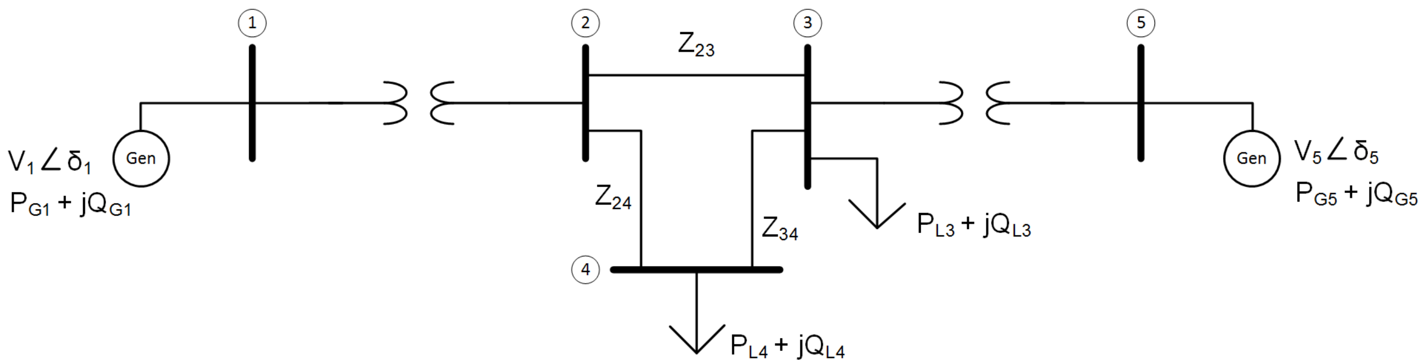
Power-Flow Analysis

The Power-Flow Problem

- A typical power system is not entirely unlike the simple circuit we just looked at
  - Sources are ***generators***
  - Nodes are the system ***buses***
  - Buses are interconnected by impedances of ***transmission lines*** and ***transformers***
- Inputs and outputs now include ***power*** (P and Q)
  - System equations are now ***nonlinear***
  - Can't simply solve $\boldsymbol{YV} = \boldsymbol{I}$
  - Must employ ***numerical***, ***iterative*** solution methods
- Power system analysis to determine bus voltages and power flows is called ***power-flow analysis*** or ***load-flow analysis***

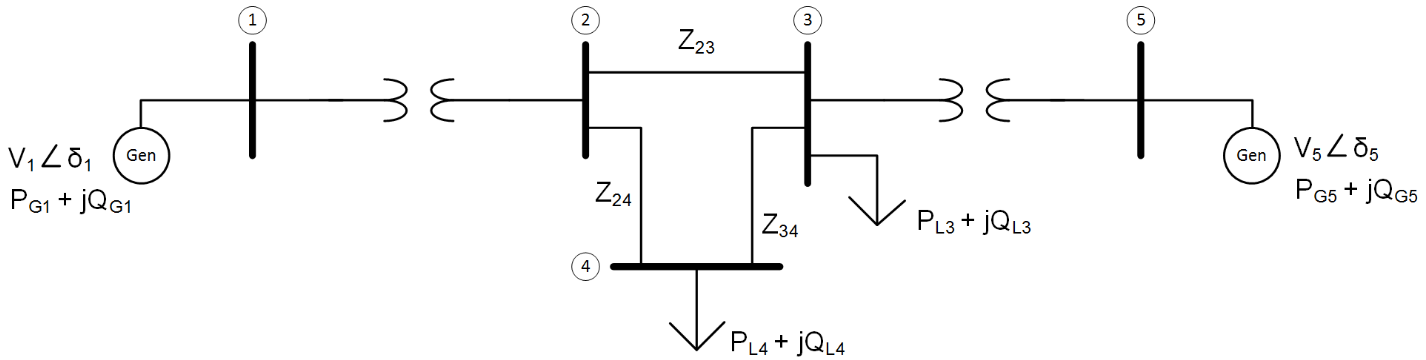
System One-Line Diagram

- Consider the one-line diagram for a simple power system



- System includes:
  - Generators
  - Buses
  - Transformers
    - Treated as equivalent circuit impedances in per-unit
  - Transmission lines
    - Equivalent circuit impedances
  - Loads

Bus Variables



- The ***buses*** are the system ***nodes***
- Four variables associated with each bus, k
  - ***Voltage magnitude***, V_k
  - ***Voltage phase angle***, δ_k
  - ***Real power*** delivered to the bus, P_k
  - ***Reactive power*** delivered to the bus, Q_k

Bus Power

- Net power delivered to bus k is the difference between power flowing from generators to bus k and power flowing from bus k to loads

$$P_k = P_{Gk} - P_{Lk}$$

$$Q_k = Q_{Gk} - Q_{Lk}$$

- Even though we've introduced power flow into the analysis, we can still write nodal equations for the system
- Voltage and current related by the ***bus admittance matrix***, $\boldsymbol{Y}_{bus}$

$$\mathbf{I} = \mathbf{Y}_{bus}\mathbf{V}$$

  - $\mathbf{Y}_{bus}$ contains the bus mutual and self admittances associated with transmission lines and transformers
  - For an N bus system, $\mathbf{V}$ is an $N \times 1$ vector of bus voltages
  - $\mathbf{I}$ is an $N \times 1$ vector of source currents flowing into each bus
    - From generators and loads

Types of Buses

- There are four variables associated with each bus
  - $V_k = |\boldsymbol{V}_k|$
  - $\delta_k = \angle \boldsymbol{V}_k$
  - P_k
  - Q_k
- Two variables are inputs to the power-flow problem
  - Known
- Two are outputs
  - To be calculated
- Buses are categorized into three types depending on which quantities are inputs and which are outputs
  - ***Slack bus*** (***swing bus***)
  - ***Load bus*** (***PQ bus***)
  - ***Voltage-controlled bus*** (***PV bus***)

Bus Types

- **Slack bus (swing bus)**:
  - Reference bus
  - Typically bus 1
  - Inputs are voltage magnitude, V_1, and phase angle, δ_1
    - Typically $1.0\angle 0°$
  - Power, P_1 and Q_1, is computed

- **Load bus (PQ bus)**:
  - Buses to which only loads are connected
  - Real power, P_k, and reactive power, Q_k, are the knowns
  - V_k and δ_k are calculated
  - Majority of power system buses are load buses

Bus Types

- **Voltage-controlled bus (PV bus)**:
  - Buses connected to generators
  - Buses with shunt reactive compensation
  - Real power, P_k, and voltage magnitude, V_k, are known inputs
  - Reactive power, Q_k, and voltage phase angle, δ_k, are calculated

Solving the Power-Flow Problem

- The power-flow solution involves determining:
  - V_k, δ_k, P_k, and Q_k
- There are N buses
  - Each with two unknown quantities
- There are $2N$ unknown quantities in total
  - Need $2N$ equations
- N of these equations are the nodal equations

$$\boldsymbol{I} = \boldsymbol{Y}\boldsymbol{V} \tag{1}$$

- The other N equations are the power-balance equations

$$\boldsymbol{S}_k = P_k + jQ_k = \boldsymbol{V}_k \boldsymbol{I}_k^* \tag{2}$$

- From (1), the nodal equation for the k^{th} bus is

$$\boldsymbol{I}_k = \sum_{n=1}^{N} Y_{kn} \boldsymbol{V}_n \tag{3}$$

Solving the Power-Flow Problem

- Substituting (3) into (2) gives

$$P_k + jQ_k = \boldsymbol{V}_k\left(\sum_{n=1}^{N} Y_{kn}\boldsymbol{V}_n\right)^* \quad (4)$$

- The bus voltages in (3) and (4) are phasors, which we can represent as

$$\boldsymbol{V}_n = V_n e^{j\delta_n} \quad \text{and} \quad \boldsymbol{V}_k = V_k e^{j\delta_k} \quad (5)$$

- The admittances can also be written in polar form

$$Y_{kn} = |Y_{kn}|e^{j\theta_{kn}} \quad (6)$$

- Using (5) and (6) in (4) gives

$$P_k + jQ_k = V_k e^{j\delta_k}\left(\sum_{n=1}^{N} |Y_{kn}|e^{j\theta_{kn}} V_n e^{j\delta_n}\right)^*$$

$$P_k + jQ_k = V_k \sum_{n=1}^{N} |Y_{kn}| V_n e^{j(\delta_k - \delta_n - \theta_{kn})} \quad (7)$$

Solving the Power-Flow Problem

- In Cartesian form, (7) becomes

$$P_k + jQ_k = V_k \sum_{n=1}^{N} |Y_{kn}| V_n \left[\cos(\delta_k - \delta_n - \theta_{kn}) + j\sin(\delta_k - \delta_n - \theta_{kn})\right] \quad (8)$$

- From (8), active power is

$$P_k = V_k \sum_{n=1}^{N} |Y_{kn}| V_n \cos(\delta_k - \delta_n - \theta_{kn}) \quad (9)$$

- And, reactive power is

$$Q_k = V_k \sum_{n=1}^{N} |Y_{kn}| V_n \sin(\delta_k - \delta_n - \theta_{kn}) \quad (10)$$

Solving the Power-Flow Problem

$$P_k = V_k \sum_{n=1}^{N} |Y_{kn}| V_n \cos(\delta_k - \delta_n - \theta_{kn}) \qquad (9)$$

$$Q_k = V_k \sum_{n=1}^{N} |Y_{kn}| V_n \sin(\delta_k - \delta_n - \theta_{kn}) \qquad (10)$$

- Solving the power-flow problem amounts to finding a solution to a system of nonlinear equations, (9) and (10)
- Must be solved using ***numerical***, ***iterative*** algorithms
  - Typically Newton-Raphson
- In practice, commercial software packages are available for power-flow analysis
  - E.g. PowerWorld, CYME, ETAP
- We'll now learn to solve the power-flow problem
  - ***Numerical***, ***iterative*** algorithm
    - ***Newton-Raphson***

Solving the Power-Flow Problem

- First, we'll introduce a variety of numerical algorithms for solving equations and systems of equations
  - Linear system of equations
    - Direct solution
      - Gaussian elimination
    - Iterative solution
      - Jacobi
      - Gauss-Seidel
  - Nonlinear equations
    - Iterative solution
      - Newton-Raphson
  - Nonlinear system of equations
    - Iterative solution
      - Newton-Raphson

Linear Systems of Equations – Direct Solution

Solving Linear Systems of Equations

- ***Gaussian elimination***
  - Direct (i.e. non-iterative) solution
  - Two parts to the algorithm:
    - Forward elimination
    - Back substitution

Gaussian Elimination

- Consider a system of equations

$$-4x_1 + 7x_3 = -5$$
$$2x_1 - 3x_2 + 5x_3 = -12$$
$$x_2 - 3x_3 = 3$$

- This can be expressed in matrix form:

$$\begin{bmatrix} -4 & 0 & 7 \\ 2 & -3 & 5 \\ 0 & 1 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -5 \\ -12 \\ 3 \end{bmatrix}$$

- In general

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{y}$$

- For a system of three equations with three unknowns:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Gaussian Elimination

- We'll use a 3×3 system as an example to develop the Gaussian elimination algorithm

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

- First, create the ***augmented system matrix***

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & \vdots & y_1 \\ A_{21} & A_{22} & A_{23} & \vdots & y_2 \\ A_{31} & A_{32} & A_{33} & \vdots & y_3 \end{bmatrix}$$

- Each row represents and equation
  - N rows for N equations
- ***Row operations*** do not affect the system
  - Multiply a row by a constant
  - Add or subtract rows from one another and replace row with the result

Gaussian Elimination – Forward Elimination

- Perform row operations to reduce the augmented matrix to ***upper triangular***
  - Only zeros below the main diagonal
  - Eliminate x_i from the $(i+1)^{\text{st}}$ through the N^{th} equations for $i = 1 \dots N$
  - ***Forward elimination***

- After forward elimination, we have

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & \vdots & y_1 \\ 0 & A'_{22} & A'_{23} & \vdots & y'_2 \\ 0 & 0 & A'_{33} & \vdots & y'_3 \end{bmatrix}$$

  - Where the *prime* notation (e.g. A'_{22}) indicates that the value has been changed from its original value

Gaussian Elimination – Back Substitution

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & \vdots & y_1 \\ 0 & A'_{22} & A'_{23} & \vdots & y'_2 \\ 0 & 0 & A'_{33} & \vdots & y'_3 \end{bmatrix}$$

- The last row represents an equation with only a single unknown

$$A'_{33} \cdot x_3 = y'_3$$

  - Solve for x_3

$$x_3 = \frac{y'_3}{A'_{33}}$$

- The second-to-last row represents an equation with two unknowns

$$A'_{22} \cdot x_2 + A'_{23} \cdot x_3 = y'_2$$

  - Substitute in newly-found value of x_3
  - Solve for x_2
- Substitute values for x_2 and x_3 into the first-row equation
  - Solve for x_1
- This process is ***back substitution***

Gaussian elimination

- Gaussian elimination summary
  - Create the augmented system matrix
  - Forward elimination
    - Reduce to an upper-triangular matrix
  - Back substitution
    - Starting with x_N, solve for x_i for $i = N \dots 1$

- A ***direct solution*** algorithm
  - Exact value for each x_i arrived at with a single execution of the algorithm
- Alternatively, we can use an iterative algorithm
  - The ***Jacobi method***

Linear Systems of Equations – Iterative Solution – Jacobi Method

Jacobi Method

- Consider a system of N linear equations

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{y}$$

$$\begin{bmatrix} A_{1,1} & \cdots & A_{1,N} \\ \vdots & \ddots & \vdots \\ A_{N,1} & \cdots & A_{N,N} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

- The k^{th} equation (k^{th} row) is

$$A_{k,1}x_1 + A_{k,2}x_2 + \cdots + A_{k,k}x_k + \cdots + A_{k,N}x_N = y_k \tag{11}$$

- Solve (11) for x_k

$$x_k = \frac{1}{A_{k,k}} [y_k - (A_{k,1}x_1 + A_{k,2}x_2 + \cdots + A_{k,k-1}x_{k-1} + \\ + A_{k,k+1}x_{k+1} + \cdots + A_{k,N}x_N)] \tag{12}$$

Jacobi Method

- Simplify (12) using summing notation

$$x_k = \frac{1}{A_{k,k}}\left[y_k - \sum_{n=1}^{k-1} A_{k,n}x_n - \sum_{n=k+1}^{N} A_{k,n}x_n\right], \qquad k = 1 \ldots N \tag{13}$$

- An equation for x_k
  - But, of course, we don't yet know all other x_n values
- Use (13) as an ***iterative expression***

$$x_{k,i+1} = \frac{1}{A_{k,k}}\left[y_k - \sum_{n=1}^{k-1} A_{k,n}x_{n,i} - \sum_{n=k+1}^{N} A_{k,n}x_{n,i}\right], \qquad k = 1 \ldots N \tag{14}$$

  - The i subscript indicates iteration number
    - $x_{k,i+1}$ is the updated value from the current iteration
    - $x_{n,i}$ is a value from the previous iteration

Jacobi Method

$$x_{k,i+1} = \frac{1}{A_{k,k}}\left[y_k - \sum_{n=1}^{k-1} A_{k,n}x_{n,i} - \sum_{n=k+1}^{N} A_{k,n}x_{n,i}\right], \qquad k = 1 \dots N \qquad (14)$$

- Old values of x_n, on the right-hand side, are used to update x_k on the left-hand side
- Start with an ***initial guess*** for all unknowns, $\mathbf{x}_0$
- Iterate until adequate ***convergence*** is achieved
  - Until a specified ***stopping criterion*** is satisfied
  - Convergence is not guaranteed

Convergence

- An approximation of $\mathbf{x}$ is refined on each iteration
- Continue to iterate until we're *close* to the right answer for the vector of unknowns, $\mathbf{x}$
  - Assume we've converged to the right answer when $\mathbf{x}$ changes very little from iteration to iteration
- On each iteration, calculate a ***relative error*** quantity

$$\varepsilon_i = \max\left(\left|\frac{x_{k,i+1} - x_{k,i}}{x_{k,i}}\right|\right), \qquad k = 1 \dots N$$

- Iterate until

$$\varepsilon_i \leq \varepsilon_s$$

where ε_s is a chosen ***stopping criterion***

Jacobi Method – Matrix Form

- The Jacobi method iterative formula, (14), can be rewritten in matrix form:

$$\mathbf{x}_{i+1} = \mathbf{M}\mathbf{x}_i + \mathbf{D}^{-1}\mathbf{y} \tag{15}$$

where $\mathbf{D}$ is the diagonal elements of $\mathbf{A}$

$$\mathbf{D} = \begin{bmatrix} A_{1,1} & 0 & \cdots & 0 \\ 0 & A_{2,2} & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \cdots & 0 & A_{N,N} \end{bmatrix}$$

and

$$\mathbf{M} = \mathbf{D}^{-1}(\mathbf{D} - \mathbf{A}) \tag{16}$$

  - Recall that the inverse of a diagonal matrix is given by inverting each diagonal element

$$\mathbf{D}^{-1} = \begin{bmatrix} 1/A_{1,1} & 0 & \cdots & 0 \\ 0 & 1/A_{2,2} & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \cdots & 0 & 1/A_{N,N} \end{bmatrix}$$

Jacobi Method – Example

- Consider the following system of equations

$$-4x_1 + 7x_3 = -5$$
$$2x_1 - 3x_2 + 5x_3 = -12$$
$$x_2 - 3x_3 = 3$$

- In matrix form:

$$\begin{bmatrix} -4 & 0 & 7 \\ 2 & -3 & 5 \\ 0 & 1 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -5 \\ -12 \\ 3 \end{bmatrix}$$

- Solve using the Jacobi method

Jacobi Method – Example

- The iteration formula is

$$\mathbf{x}_{i+1} = \mathbf{M}\mathbf{x}_i + \mathbf{D}^{-1}\mathbf{y}$$

where

$$\mathbf{D} = \begin{bmatrix} -4 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & -3 \end{bmatrix} \qquad \mathbf{D}^{-1} = \begin{bmatrix} -0.25 & 0 & 0 \\ 0 & -0.333 & 0 \\ 0 & 0 & -0.333 \end{bmatrix}$$

$$\mathbf{M} = \mathbf{D}^{-1}(\mathbf{D} - \mathbf{A}) = \begin{bmatrix} 0 & 0 & 1.75 \\ 0.667 & 0 & 1.667 \\ 0 & 0.333 & 0 \end{bmatrix}$$

- To begin iteration, we need a starting point
  - Initial guess for unknown values, $\mathbf{x}$
  - Often, we have some idea of the answer
  - Here, arbitrarily choose

$$\mathbf{x}_0 = [10 \quad 25 \quad 10]^T$$

Jacobi Method – Example

- At each iteration, calculate

$$\mathbf{x}_{i+1} = \mathbf{M}\mathbf{x}_i + \mathbf{D}^{-1}\mathbf{y}$$

$$\begin{bmatrix} x_{1,i+1} \\ x_{2,i+1} \\ x_{3,i+1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1.75 \\ 0.667 & 0 & 1.667 \\ 0 & 0.333 & 0 \end{bmatrix} \begin{bmatrix} x_{1,i} \\ x_{2,i} \\ x_{3,i} \end{bmatrix} + \begin{bmatrix} 1.25 \\ 4 \\ -1 \end{bmatrix}$$

- For $i = 1$:

$$\mathbf{x}_1 = \begin{bmatrix} x_{1,1} \\ x_{2,1} \\ x_{3,1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1.75 \\ 0.667 & 0 & 1.667 \\ 0 & 0.333 & 0 \end{bmatrix} \begin{bmatrix} 10 \\ 25 \\ 10 \end{bmatrix} + \begin{bmatrix} 1.25 \\ 4 \\ -1 \end{bmatrix}$$

$$\mathbf{x}_1 = [18.75 \quad 27.33 \quad 7.33]^T$$

- The relative error is

$$\varepsilon_1 = \max\left(\left|\frac{x_{k,1} - x_{k,0}}{x_{k,0}}\right|\right) = 0.875$$

Jacobi Method – Example

- For $i = 2$:

$$\mathbf{x}_2 = \begin{bmatrix} x_{1,2} \\ x_{2,2} \\ x_{3,2} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1.75 \\ 0.667 & 0 & 1.667 \\ 0 & 0.333 & 0 \end{bmatrix} \begin{bmatrix} 18.75 \\ 27.33 \\ 7.33 \end{bmatrix} + \begin{bmatrix} 1.25 \\ 4 \\ -1 \end{bmatrix}$$

$$\mathbf{x}_2 = [14.08 \quad 28.72 \quad 8.11]^T$$

- The relative error is

$$\varepsilon_2 = \max\left(\left|\frac{x_{k,2} - x_{k,1}}{x_{k,1}}\right|\right) = 0.249$$

- Continue to iterate until relative error falls below a specified stopping condition

Jacobi Method – Example

- Automate with computer code, e.g. MATLAB
- Setup the system of equations

```
% coefficient matrix
A = [-4,0,7;2,-3,5;0,1,-3];

% vector of knowns
y = [-5;-12;3];
```

- Initialize matrices and parameters for iteration

```
reltol = 1e-6;
eps = 1;

max_iter = 600;
iter = 0;

% initial guess for x
x = [10;25;10];

D = diag(diag(A));
invD = inv(D);

M = invD*(D - A);
```

Jacobi Method – Example

- Loop to continue iteration as long as:
  - Stopping criterion is not satisfied
  - Maximum number of iterations is not exceeded

```
while((eps > reltol) && (iter < max_iter))
        xold = x;
        x = M*xold + invD*y;

        eps = max(abs((x - xold)./xold));

        iter = iter + 1;
end
```

- On each iteration
  - Use previous $\mathbf{x}$ values to update $\mathbf{x}$
  - Calculate relative error
  - Increment the number of iterations

Jacobi Method – Example

- Set $\varepsilon_s = 1 \times 10^{-6}$ and iterate:

i	$\mathbf{x}_i$	ε_i
0	$[10 \quad 25 \quad 10]^T$	-
1	$[18.75 \quad 27.33 \quad 7.33]^T$	0.875
2	$[14.08 \quad 28.72 \quad 8.11]^T$	0.249
3	$[15.44 \quad 26.91 \quad 8.57]^T$	0.097
4	$[16.25 \quad 28.59 \quad 7.97]^T$	0.071
5	$[15.20 \quad 28.12 \quad 8.53]^T$	0.070
6	$[16.18 \quad 28.35 \quad 8.37]^T$	0.065
$\vdots$	$\vdots$	$\vdots$
371	$[20.50 \quad 36.00 \quad 11.00]^T$	0.995×10^{-6}

- Convergence achieved in 371 iterations

Linear Systems of Equations – Iterative Solution – Gauss-Seidel

Gauss-Seidel Method

- The iterative formula for the Jacobi method is

$$x_{k,i+1} = \frac{1}{A_{k,k}}\left[y_k - \sum_{n=1}^{k-1} A_{k,n}x_{n,i} - \sum_{n=k+1}^{N} A_{k,n}x_{n,i}\right], \qquad k = 1 \dots N \tag{14}$$

- Note that only old values of x_n (i.e. $x_{n,i}$) are used to update the value of x_k
- Assume the $x_{k,i+1}$ values are determined in order of increasing k
  - When updating $x_{k,i+1}$, all $x_{n,i+1}$ values are already known for $n < k$
  - We can use those updated values to calculate $x_{k,i+1}$
  - The ***Gauss-Seidel method***

Gauss-Seidel Method

- Now use the x_n values already updated on the current iteration to update x_k
  - That is, $x_{n,i+1}$ for $n < k$
- ***Gauss-Seidel*** iterative formula

$$x_{k,i+1} = \frac{1}{A_{k,k}}\left[y_k - \sum_{n=1}^{k-1} A_{k,n}x_{n,i+1} - \sum_{n=k+1}^{N} A_{k,n}x_{n,i}\right], \qquad k = 1 \ldots N \tag{17}$$

- Note that only the first summation has changed
  - For already updated x values
  - x_n for $n < k$
  - Number of already-updated values used depends on k

Gauss-Seidel – Matrix Form

- In matrix form the iterative formula is the same as for the Jacobi method

$$\mathbf{x}_{i+1} = \mathbf{M}\mathbf{x}_i + \mathbf{D}^{-1}\mathbf{y} \tag{15}$$

where, again

$$\mathbf{M} = \mathbf{D}^{-1}(\mathbf{D} - \mathbf{A}) \tag{16}$$

but now $\mathbf{D}$ is the lower triangular part of $\mathbf{A}$

$$\mathbf{D} = \begin{bmatrix} A_{1,1} & 0 & \cdots & 0 \\ A_{2,1} & A_{2,2} & 0 & \vdots \\ \vdots & \vdots & \ddots & 0 \\ A_{N,1} & A_{N,2} & \cdots & A_{N,N} \end{bmatrix}$$

- Otherwise, the algorithm and computer code is identical to that of the Jacobi method

Gauss-Seidel – Example

- Apply Gauss-Seidel to our previous example
  - $x_0 = [10 \quad 25 \quad 10]^T$
  - $\varepsilon_s = 1 \times 10^{-6}$

i	$\mathbf{x}_i$	ε_i
0	$[10 \quad 25 \quad 10]^T$	-
1	$[18.75 \quad 33.17 \quad 10.06]^T$	0.875
2	$[18.85 \quad 33.32 \quad 10.11]^T$	0.005
3	$[18.94 \quad 33.47 \quad 10.16]^T$	0.005
4	$[19.03 \quad 33.61 \quad 10.20]^T$	0.005
$\vdots$	$\vdots$	$\vdots$
151	$[20.50 \quad 36.00 \quad 11.00]^T$	0.995×10^{-6}

- Convergence achieved in 151 iterations
  - Compared to 371 for the Jacobi method

Nonlinear Equations

Nonlinear Equations

- Solution methods we've seen so far work only for ***linear*** equations
- Now, we introduce an iterative method for solving a ***single nonlinear equation***
  - ***Newton-Raphson*** method
- Next, we'll apply the Newton-Raphson method to a ***system of nonlinear equations***
- Finally, we'll use Newton-Raphson to solve the ***power-flow problem***

Newton-Raphson Method

- Want to solve

$$y = f(x)$$

where $f(x)$ is a nonlinear function

- That is, we want to find x, given a known nonlinear function, f, and a known output, y

- ***Newton-Raphson method***
  - Based on a ***first-order Taylor series approximation*** to $f(x)$
  - The ***nonlinear*** $f(x)$ is approximated as ***linear*** to update our approximation to the solution, x, on each iteration

Taylor Series Approximation

- Taylor series approximation
  - Given:
    - A function, $f(x)$
    - Value of the function at some value of x, $f(x_0)$
  - Approximate:
    - Value of the function at some other value of x

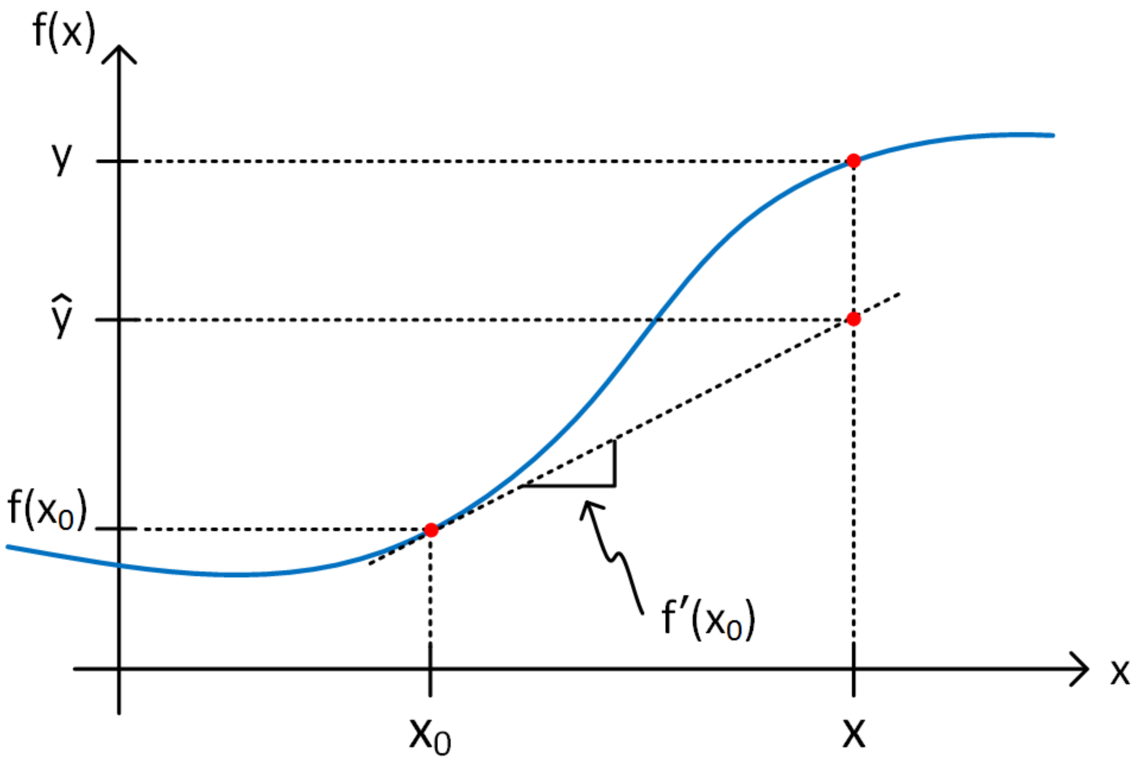
- ***First-order Taylor series approximation***
  - Approximate $f(x)$ using only its first derivative
  - $f(x)$ approximated as linear – constant slope

$$y = f(x) \approx f(x_0) + \left.\frac{df}{dx}\right|_{x=x_0} (x - x_0) = \hat{y}$$

First-Order Taylor Series Approximation

- Approximate value of the function at x

$$f(x) \approx \hat{y} = f(x_0) + f'(x_0)(x - x_0)$$



Newton-Raphson Method

- First order Taylor series approximation is

$$y \approx f(x_0) + f'(x_0)(x - x_0)$$

- Letting this be an equality and rearranging gives an ***iterative formula*** for updating an approximation to x

$$y = f(x_i) + f'(x_i)(x_{i+1} - x_i)$$

$$f'(x_i)(x_{i+1} - x_i) = y - f(x_i)$$

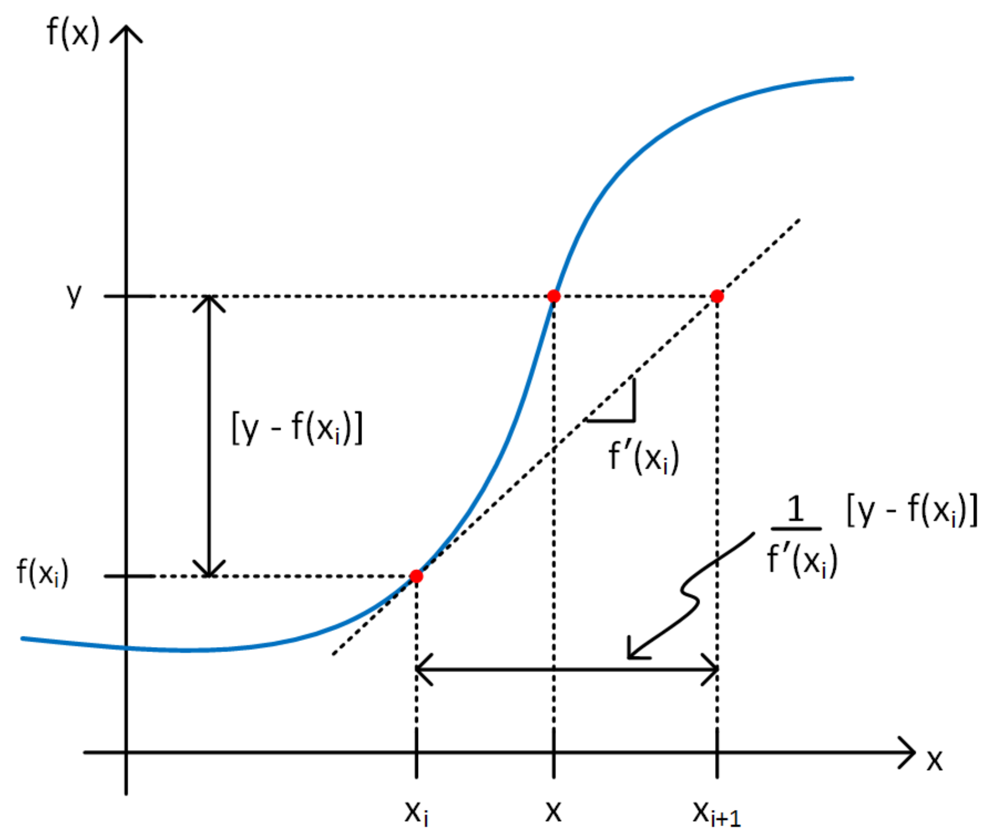
$$x_{i+1} = x_i + \frac{1}{f'(x_i)}[y - f(x_i)] \qquad (18)$$

- Initialize with a best guess at x, x_0
- Iterate (18) until
  - A stopping criterion is satisfied, or
  - The maximum number of iterations is reached

First-Order Taylor Series Approximation

$$x_{i+1} = x_i + \frac{1}{f'(x_i)}[y - f(x_i)]$$

- Now using the Taylor series approximation in a different way
  - Not approximating the value of y = f(x) at x, but, instead
  - Approximating the value of x where f(x) = y



Newton-Raphson – Example

- Consider the following nonlinear equation

$$y = f(x) = x^3 + 10 = 20$$

- Apply Newton-Raphson to solve
  - Find x, such that $y = f(x) = 20$
- The derivative function is

$$f'(x) = 3x^2$$

- Initial guess for x

$$x_0 = 1$$

- Iterate using the formula given by (18)

Newton-Raphson – Example

- $\underline{i = 1}$:

$$x_1 = x_0 + f'(x_0)^{-1}[y - f(x_0)]$$

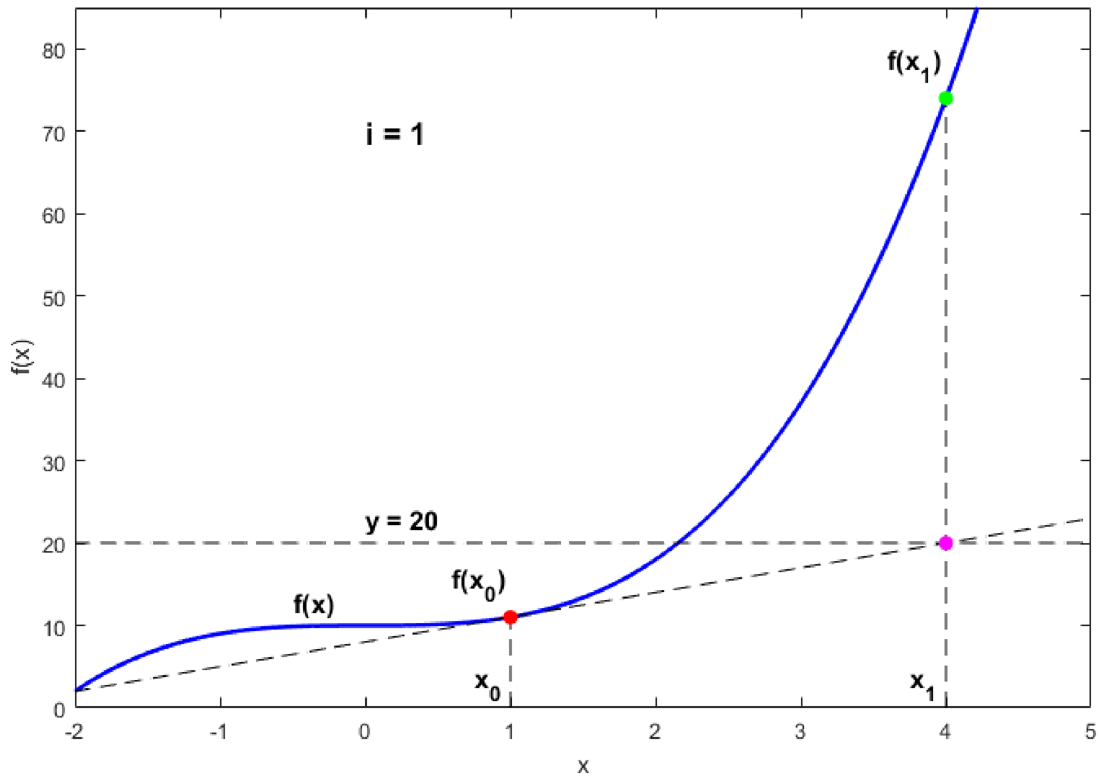
$$x_1 = 1 + [3 \cdot 1^2]^{-1}[20 - (1^3 + 10)]$$

$$x_1 = 4$$

$$\varepsilon_1 = \left|\frac{x_1 - x_0}{x_0}\right|$$

$$\varepsilon_1 = \left|\frac{4 - 1}{1}\right| = 3$$

$$\boxed{x_1 = 4,\ \varepsilon_1 = 3}$$



Newton-Raphson – Example

- $\underline{i = 2}$:

$$x_2 = x_1 + f'(x_1)^{-1}[y - f(x_1)]$$

$$x_2 = 4 + [3 \cdot 4^2]^{-1}[20 - (4^3 + 10)]$$

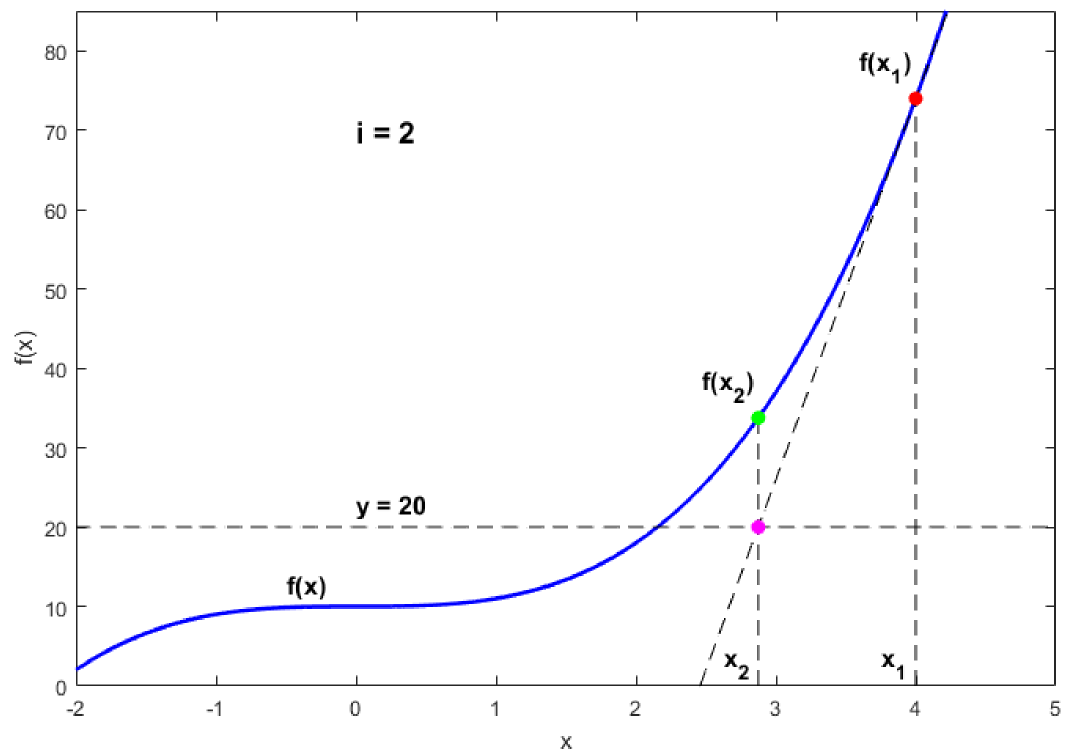
$$x_2 = 2.875$$

$$\varepsilon_2 = \left|\frac{x_2 - x_1}{x_1}\right|$$

$$\varepsilon_2 = \left|\frac{2.875 - 4}{4}\right|$$

$$\varepsilon_2 = 0.281$$

$$\boxed{x_2 = 2.875,\ \varepsilon_2 = 0.281}$$



Newton-Raphson – Example

- $\underline{i = 3}$:

$$x_3 = x_2 + f'(x_2)^{-1}[y - f(x_2)]$$

$$x_3 = 2.875 + [3 \cdot 2.875^2]^{-1}[20 - (2.875^3 + 10)]$$

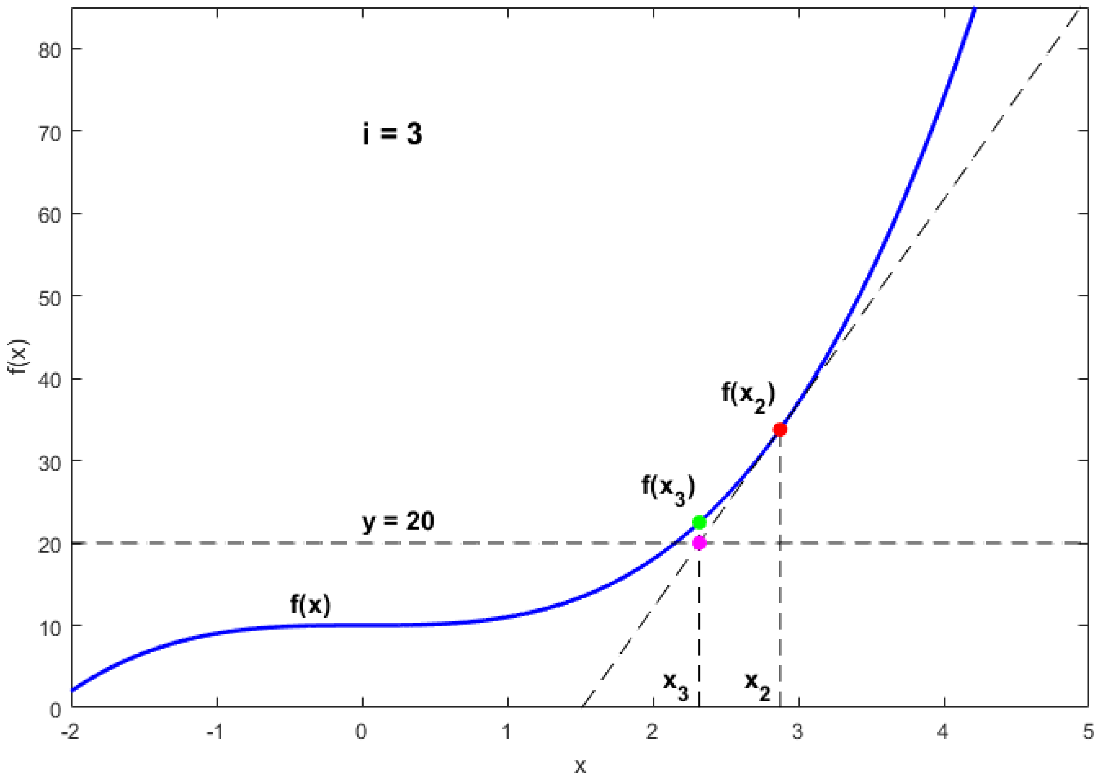
$$x_3 = 2.32$$

$$\varepsilon_3 = \left|\frac{x_3 - x_2}{x_2}\right|$$

$$\varepsilon_3 = \left|\frac{2.32 - 2.875}{2.875}\right|$$

$$\varepsilon_3 = 0.193$$

$$\boxed{x_3 = 2.32,\ \varepsilon_3 = 0.193}$$



Newton-Raphson – Example

- $\underline{i = 4}$:

$$x_4 = 2.166, \ \varepsilon_4 = 0.066$$

- $\underline{i = 5}$:

$$x_5 = 2.155, \ \varepsilon_5 = 0.005$$

- $\underline{i = 6}$:

$$x_6 = 2.154, \ \varepsilon_6 = 28.4 \times 10^{-6}$$

- $\underline{i = 7}$:

$$x_7 = 2.154, \ \varepsilon_7 = 0.808 \times 10^{-9}$$

- Convergence achieved very quickly
- Next, we'll see how to apply Newton-Raphson to a ***system*** of nonlinear equations

Example Problems

Perform three iterations toward the solution of the following system of equations using the ***Jacobi*** method. Let $\mathbf{x}_0 = [0, 0]^T$.

$$2x_1 + x_2 = 12$$

$$2x_1 + 3x_2 = 5$$

Perform three iterations toward the solution of the following system of equations using the ***Gauss-Seidel*** method. Let $\mathbf{x}_0 = [0, 0]^T$.

$$2x_1 + x_2 = 12$$

$$2x_1 + 3x_2 = 5$$

Perform three iterations toward the solution of the following equation using the Newton-Raphson method. Let $\mathbf{x}_0 = 0$.

$$f(x) = \cos(x) + 3x = 10$$

Nonlinear Systems of Equations

Nonlinear Systems of Equations

- Now, consider a system of nonlinear equations
  - Can be represented as a vector of N functions
  - Each is a function of an N-vector of unknown variables

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(x_1, x_2, \cdots, x_N) \\ f_2(x_1, x_2, \cdots, x_N) \\ \vdots \\ f_N(x_1, x_2, \cdots, x_N) \end{bmatrix}$$

- We can again approximate this function using a first-order Taylor series

$$\mathbf{y} = \mathbf{f}(\mathbf{x}) \approx \mathbf{f}(\mathbf{x}_0) + \mathbf{f}'(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) \tag{19}$$

  - Note that all variables are N-vectors
    - $\mathbf{f}$ is an N-vector of known, nonlinear functions
    - $\mathbf{x}$ is an N-vector of unknown values – this is what we want to solve for
    - $\mathbf{y}$ is an N-vector of known values
    - $\mathbf{x_0}$ is an N-vector of $\mathbf{x}$ values for which $\mathbf{f}(\mathbf{x}_0)$ is known

Newton-Raphson Method

- Equation (19) is the basis for our Newton-Raphson iterative formula
  - Again, let it be an equality and solve for $\mathbf{x}$

$$\mathbf{y} - \mathbf{f}(\mathbf{x}_0) = \mathbf{f}'(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)$$

$$[\mathbf{f}'(\mathbf{x}_0)]^{-\mathbf{1}}[\mathbf{y} - \mathbf{f}(\mathbf{x}_0)] = \mathbf{x} - \mathbf{x}_0$$

$$\mathbf{x} = \mathbf{x}_0 + [\mathbf{f}'(\mathbf{x}_0)]^{-\mathbf{1}}[\mathbf{y} - \mathbf{f}(\mathbf{x}_0)]$$

- This last expression can be used as an iterative formula

$$\mathbf{x}_{i+1} = \mathbf{x}_i + [\mathbf{f}'(\mathbf{x}_i)]^{-\mathbf{1}}[\mathbf{y} - \mathbf{f}(\mathbf{x}_i)]$$

- The derivative term on the right-hand side of (20) is an $N \times N$ matrix
  - The ***Jacobian*** matrix, $\mathbf{J}$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{J}_i^{-\mathbf{1}}[\mathbf{y} - \mathbf{f}(\mathbf{x}_i)] \tag{20}$$

The Jacobian Matrix

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{J}_i^{-1}[\mathbf{y} - \mathbf{f}(\mathbf{x}_i)] \tag{20}$$

- ***Jacobian matrix***
  - $N \times N$ matrix of partial derivatives for $\mathbf{f}(\mathbf{x})$
  - Evaluated at the current value of $\mathbf{x}$, $\mathbf{x}_i$

$$\mathbf{J}_i = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_N}{\partial x_1} & \frac{\partial f_N}{\partial x_2} & \cdots & \frac{\partial f_N}{\partial x_N} \end{bmatrix}_{\mathbf{x}=\mathbf{x}_i}$$

Newton-Raphson Method

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{J}_i^{-1}[\mathbf{y} - \mathbf{f}(\mathbf{x}_i)] \tag{20}$$

- We could iterate (20) until convergence or a maximum number of iterations is reached
  - Requires ***inversion*** of the Jacobian matrix
    - Computationally expensive and error prone
- Instead, go back to the Taylor series approximation

$$\mathbf{y} = \mathbf{f}(\mathbf{x}_i) + \mathbf{J}_i(\mathbf{x}_{i+1} - \mathbf{x}_i)$$

$$\mathbf{y} - \mathbf{f}(\mathbf{x}_i) = \mathbf{J}_i(\mathbf{x}_{i+1} - \mathbf{x}_i) \tag{21}$$

  - Left side of (21) represents a difference between the known and approximated outputs
  - Right side represents an increment of the approximation for $\mathbf{x}$

$$\Delta\mathbf{y}_i = \mathbf{J}_i\Delta\mathbf{x}_i \tag{22}$$

Newton-Raphson Method

$$\Delta \mathbf{y}_i = \mathbf{J}_i \Delta \mathbf{x}_i \tag{22}$$

- On each iteration:
  - Compute $\Delta \mathbf{y}_i$ and $\mathbf{J}_i$
  - Solve for $\Delta \mathbf{x}_i$ using ***Gaussian elimination***
    - Matrix inversion not required
    - Computationally robust
  - Update $\mathbf{x}$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta \mathbf{x}_i \tag{23}$$

Newton-Raphson – Example

- Apply Newton-Raphson to solve the following system of nonlinear equations

$$\mathbf{f}(\mathbf{x}) = \mathbf{y}$$

$$\begin{bmatrix} x_1^2 + 3x_2 \\ x_1 x_2 \end{bmatrix} = \begin{bmatrix} 21 \\ 12 \end{bmatrix}$$

  - Initial condition: $\mathbf{x}_0 = [1 \quad 2]^T$
  - Stopping criterion: $\varepsilon_s = 1 \times 10^{-6}$
  - Jacobian matrix

$$\mathbf{J}_i = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix}_{\mathbf{x}=\mathbf{x}_i} = \begin{bmatrix} 2x_{1,i} & 3 \\ x_{2,i} & x_{1,i} \end{bmatrix}$$

Newton-Raphson – Example

$$\Delta\mathbf{y}_i = \mathbf{J}_i \Delta\mathbf{x}_i \quad (22)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta\mathbf{x}_i \quad (23)$$

- Adjusting the indexing, we can equivalently write (22) and (23) as:

$$\Delta\mathbf{y}_{i-1} = \mathbf{J}_{i-1} \Delta\mathbf{x}_{i-1} \quad (22)$$

$$\mathbf{x}_i = \mathbf{x}_{i-1} + \Delta\mathbf{x}_{i-1} \quad (23)$$

- For iteration i:
  - Compute $\Delta\mathbf{y}_{i-1}$ and $\mathbf{J}_{i-1}$
  - Solve (22) for $\Delta\mathbf{x}_{i-1}$
  - Update $\mathbf{x}$ using (23)

Newton-Raphson – Example

- $\underline{i = 1}$:

$$\Delta y_0 = \mathbf{y} - \mathbf{f}(\mathbf{x}_0) = \begin{bmatrix} 21 \\ 12 \end{bmatrix} - \begin{bmatrix} 7 \\ 2 \end{bmatrix} = \begin{bmatrix} 14 \\ 10 \end{bmatrix}$$

$$\mathbf{J}_0 = \begin{bmatrix} 2x_{1,0} & 3 \\ x_{2,0} & x_{1,0} \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix}$$

$$\Delta \mathbf{x}_0 = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

$$\mathbf{x}_1 = \mathbf{x}_0 + \Delta \mathbf{x}_0 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 4 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \end{bmatrix}$$

$$\varepsilon_1 = \max\left(\left|\frac{x_{k,1} - x_{k,0}}{x_{k,0}}\right|\right), \qquad k = 1 \ldots N$$

$$\boxed{x_1 = \begin{bmatrix} 5 \\ 4 \end{bmatrix}, \quad \varepsilon_1 = 4}$$

Newton-Raphson – Example

- $\underline{i = 2}$:

$$\Delta y_1 = \mathbf{y} - \mathbf{f}(\mathbf{x}_1) = \begin{bmatrix} 21 \\ 12 \end{bmatrix} - \begin{bmatrix} 37 \\ 20 \end{bmatrix} = \begin{bmatrix} -16 \\ -8 \end{bmatrix}$$

$$\mathbf{J}_1 = \begin{bmatrix} 2x_{1,1} & 3 \\ x_{2,1} & x_{1,1} \end{bmatrix} = \begin{bmatrix} 10 & 3 \\ 4 & 5 \end{bmatrix}$$

$$\Delta\mathbf{x}_1 = \begin{bmatrix} -1.474 \\ -0.421 \end{bmatrix}$$

$$\mathbf{x}_2 = \mathbf{x}_1 + \Delta\mathbf{x}_1 = \begin{bmatrix} 5 \\ 4 \end{bmatrix} + \begin{bmatrix} -1.474 \\ -0.421 \end{bmatrix} = \begin{bmatrix} 3.526 \\ 3.579 \end{bmatrix}$$

$$\varepsilon_2 = \max\left(\left|\frac{x_{k,2} - x_{k,1}}{x_{k,1}}\right|\right), \qquad k = 1 \dots N$$

$$\boxed{x_2 = \begin{bmatrix} 3.526 \\ 3.579 \end{bmatrix}, \quad \varepsilon_2 = 0.295}$$

Newton-Raphson – Example

- $\underline{i = 3}$:

$$\Delta y_2 = \mathbf{y} - \mathbf{f}(\mathbf{x}_2) = \begin{bmatrix} 21 \\ 12 \end{bmatrix} - \begin{bmatrix} 23.172 \\ 12.621 \end{bmatrix} = \begin{bmatrix} -2.172 \\ -0.621 \end{bmatrix}$$

$$\mathbf{J}_2 = \begin{bmatrix} 2x_{1,2} & 3 \\ x_{2,2} & x_{1,2} \end{bmatrix} = \begin{bmatrix} 7.053 & 3 \\ 3.579 & 3.526 \end{bmatrix}$$

$$\Delta \mathbf{x}_2 = \begin{bmatrix} -0.410 \\ 0.240 \end{bmatrix}$$

$$\mathbf{x}_3 = \mathbf{x}_2 + \Delta \mathbf{x}_2 = \begin{bmatrix} 3.526 \\ 3.579 \end{bmatrix} + \begin{bmatrix} -0.410 \\ 0.240 \end{bmatrix} = \begin{bmatrix} 3.116 \\ 3.819 \end{bmatrix}$$

$$\varepsilon_3 = \max\left(\left|\frac{x_{k,3} - x_{k,2}}{x_{k,2}}\right|\right), \qquad k = 1 \ldots N$$

$$\boxed{x_3 = \begin{bmatrix} 3.116 \\ 3.819 \end{bmatrix}, \quad \varepsilon_3 = 0.116}$$

Newton-Raphson – Example

- $\underline{i = 7}$:

$$\Delta y_6 = \mathbf{y} - \mathbf{f}(\mathbf{x}_6) = \begin{bmatrix} 21 \\ 12 \end{bmatrix} - \begin{bmatrix} 21.000 \\ 12.000 \end{bmatrix} = \begin{bmatrix} -0.527 \times 10^{-7} \\ 0.926 \times 10^{-7} \end{bmatrix}$$

$$\mathbf{J}_6 = \begin{bmatrix} 2x_{1,6} & 3 \\ x_{2,6} & x_{1,6} \end{bmatrix} = \begin{bmatrix} 6.000 & 3 \\ 4.000 & 3.000 \end{bmatrix}$$

$$\Delta \mathbf{x}_6 = \begin{bmatrix} -0.073 \times 10^{-6} \\ 0.128 \times 10^{-6} \end{bmatrix}$$

$$\mathbf{x}_7 = \mathbf{x}_6 + \Delta \mathbf{x}_6 = \begin{bmatrix} 3.000 \\ 4.000 \end{bmatrix} + \begin{bmatrix} -0.073 \times 10^{-6} \\ 0.128 \times 10^{-6} \end{bmatrix} = \begin{bmatrix} 3.000 \\ 4.000 \end{bmatrix}$$

$$\varepsilon_7 = \max\left(\left|\frac{x_{k,7} - x_{k,6}}{x_{k,6}}\right|\right), \qquad k = 1 \ldots N$$

$$\boxed{x_7 = \begin{bmatrix} 3.000 \\ 4.000 \end{bmatrix}, \quad \varepsilon_7 = 31.9 \times 10^{-9}}$$

Newton-Raphson – MATLAB Code

- Define the system of equations

```
f = @(x) [x(1)^2 + 3*x(2); x(1)*x(2)];
y = [21;12];
```

- Initialize **x**

```
x0 = [1;2];
x = x0;
```

- Set up solution parameters

```
reltol = 1e-6;
max_iter = 1000;
eps = 1;
iter = 0;
```

Newton-Raphson – MATLAB Code

- Iterate:
  - Compute $\Delta\mathbf{y}_{i-1}$ and $\mathbf{J}_{i-1}$
  - Solve for $\Delta\mathbf{x}_{i-1}$
  - Update $\mathbf{x}$

```
while(iter < max_iter)&&(eps > reltol)
        iter = iter + 1;

        J = [2*x(1), 3; x(2), x(1)];
        x_old = x;

%       calculate output error term
        Dy = y - f(x_old);

%       use Gaussian elimination to solve for increment to x
        Dx = J\Dy;
        x = x_old + Dx;

        eps = max(abs((x - x_old)./x_old));
end
```

Example Problems

Perform three iterations toward the solution of the following system of equations using the Newton-Raphson method. Let $\mathbf{x}_0 = [1, 1]^T$.

$$10x_1^2 + x_2 = 20$$
$$e^{x_1} - x_2 = 10$$

Power-Flow Solution – Overview

Solving the Power-Flow Problem - Overview

- Consider an N-bus power-flow problem
  - 1 slack bus
  - n_{PV} PV buses
  - n_{PQ} PQ buses

$$N = n_{PV} + n_{PQ} + 1$$

- Each bus has two unknown quantities
  - Two of V_k, δ_k, P_k, and Q_k
- For the N-R power-flow problem, V_k and δ_k are the unknown quantities
  - These are the inputs to the nonlinear system of equations – the P_k and Q_k equations – of (9) and (10)
  - Finding unknown V_k and δ_k values allows us to determine unknown P_k and Q_k values

Solving the Power-Flow Problem - Overview

- The nonlinear system of equations is

$$\mathbf{y} = \mathbf{f}(\mathbf{x})$$

- The ***unknowns***, $\mathbf{x}$, are ***bus voltages***
  - Unknown phase angles from PV and PQ buses
  - Unknown magnitudes from PQ bus

$$\mathbf{x} = \begin{bmatrix} \boldsymbol{\delta} \\ \mathbf{V} \end{bmatrix} = \begin{array}{l} \left.\begin{bmatrix} \delta_2 \\ \vdots \\ \delta_{n_{PV}+n_{PQ}+1} \\ \hline V_{n_{PV}+2} \\ \vdots \\ V_{n_{PV}+n_{PQ}+1} \end{bmatrix}\right. \end{array} \begin{array}{l} \left.\vphantom{\begin{matrix} \delta_2 \\ \vdots \\ \delta \end{matrix}}\right\} n_{PV} + n_{PQ} \\ \left.\vphantom{\begin{matrix} V \\ \vdots \\ V \end{matrix}}\right\} n_{PQ} \end{array} \tag{24}$$

Solving the Power-Flow Problem - Overview

$$\mathbf{y} = \mathbf{f}(\mathbf{x})$$

- The ***knowns*** , $\mathbf{y}$, are ***bus powers***
  - Known real power from PV and PQ buses
  - Known reactive power from PQ bus

$$\mathbf{y} = \begin{bmatrix} \mathbf{P} \\ \mathbf{Q} \end{bmatrix} = \begin{bmatrix} P_2 \\ \vdots \\ P_{n_{PV}+n_{PQ}+1} \\ \hline Q_{n_{PV}+2} \\ \vdots \\ Q_{n_{PV}+n_{PQ}+1} \end{bmatrix} \begin{matrix} n_{PV} + n_{PQ} \\ \\ \\ n_{PQ} \end{matrix} \qquad (25)$$

Solving the Power-Flow Problem - Overview

$$\mathbf{y} = \mathbf{f}(\mathbf{x})$$

- The ***system of equations*** , $\mathbf{f}$, consists of the nonlinear functions for $\mathbf{P}$ and $\mathbf{Q}$
  - Nonlinear functions of $\mathbf{V}$ and $\boldsymbol{\delta}$

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} \mathbf{P}(\mathbf{x}) \\ \mathbf{Q}(\mathbf{x}) \end{bmatrix} = \begin{array}{c} \left[\begin{array}{c} P_2(\mathbf{x}) \\ \vdots \\ \vdots \\ \hline Q_{n_{PV}+2}(\mathbf{x}) \\ \vdots \\ \vdots \end{array} \right] \end{array} \begin{array}{l} \left.\vphantom{\begin{array}{c} P \\ \vdots \\ \vdots \end{array}}\right\} n_{PV} + n_{PQ} \\ \left.\vphantom{\begin{array}{c} Q \\ \vdots \\ \vdots \end{array}}\right\} n_{PQ} \end{array} \tag{26}$$

Solving the Power-Flow Problem - Overview

- $P_k(\mathbf{x})$ and $Q_k(\mathbf{x})$ are given by

$$P_k = V_k \sum_{n=1}^{N} |Y_{kn}| V_n \cos(\delta_k - \delta_n - \theta_{kn}) \tag{9}$$

$$Q_k = V_k \sum_{n=1}^{N} |Y_{kn}| V_n \sin(\delta_k - \delta_n - \theta_{kn}) \tag{10}$$

  - Admittance matrix terms are

$$Y_{kn} = |Y_{kn}| \angle \theta_{kn}$$

Solving the Power-Flow Problem - Overview

- The iterative N-R formula is

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta\mathbf{x}_i$$

  - The increment term, $\Delta\mathbf{x}_i$, is computed through Gaussian elimination of

$$\Delta\mathbf{y}_i = \mathbf{J}_i \Delta\mathbf{x}_i$$

  - The Jacobian, J_i, is computed on each iteration
  - The ***power mismatch*** vector is

$$\Delta\mathbf{y}_i = \mathbf{y} - \mathbf{f}(\mathbf{x}_i)$$

    - $\mathbf{y}$ is the vector of known powers, as given in (25)
    - $\mathbf{f}(\mathbf{x}_i)$ are the P and Q equations given by (9) and (10)

Power-Flow Solution – Procedure

Solving the Power-Flow Problem - Procedure

- The following procedure shows how to set up and solve the power-flow problem using the N-R algorithm

---

1. Order and number buses
   - Slack bus is #1
   - Group all PV buses together next
   - Group all PQ buses together last
2. Generate the bus admittance matrix, $\mathbf{Y}$
   - And magnitude, $\mathrm{Y} = |\mathbf{Y}|$, and angle, $\theta = \angle\mathbf{Y}$, matrices

Solving the Power-Flow Problem - Procedure

3. Initialize *known* quantities
   - Slack bus: V_1 and δ_1
   - PV buses: V_k and P_k
   - PQ buses: P_k and Q_k
   - Output vector:

$$\mathbf{y} = \begin{bmatrix} \mathbf{P} \\ \mathbf{Q} \end{bmatrix}$$

4. Initialize *unknown* quantities

$$\mathbf{x}_o = \begin{bmatrix} \boldsymbol{\delta}_0 \\ \mathbf{V}_0 \end{bmatrix} = \begin{array}{c} \left[\begin{array}{c} 0 \\ \vdots \\ 0 \\ \hline 1.0 \\ \vdots \\ 1.0 \end{array}\right] \end{array} \begin{array}{l} \left.\vphantom{\begin{array}{c} 0 \\ \vdots \\ 0 \end{array}}\right\} n_{PV} + n_{PQ} \\ \left.\vphantom{\begin{array}{c} 1.0 \\ \vdots \\ 1.0 \end{array}}\right\} n_{PQ} \end{array} \qquad (24)$$

Solving the Power-Flow Problem - Procedure

5. Set up Newton-Raphson parameters
   - Tolerance for convergence, *reltol*
   - Maximum # of iterations, *max_iter*
   - Initialize relative error: $\varepsilon_0 >$ *reltol*, e.g. $\varepsilon_0 = 10$
   - Initialize iteration counter: $i = 0$
6. while ($\varepsilon >$ *reltol*) && ($i <$ *max_iter*)
   - Update ***bus voltage phasor vector***, $\mathbf{V}_i$, using magnitude and phase values from $\mathbf{x}_i$ and from knowns
   - Calculate the current injected into each bus, a vector of phasors

$$\mathbf{I}_i = \mathbf{Y} \cdot \mathbf{V}_i$$

Solving the Power-Flow Problem - Procedure

6. while ($\varepsilon > reltol$) && ($i < max\_iter$) – cont'd
   - Calculate complex, real, and reactive power injected into each bus
     - This can be done using $\mathbf{V}_i$ and $\mathbf{I}_i$ vectors and element-by-element multiplication (the .* operator in MATLAB)

$$\mathbf{S}_{k,i} = \mathbf{V}_{k,i} \cdot \mathbf{I}_{k,i}^*$$

$$P_{k,i} = Re\{\mathbf{S}_{k,i}\}$$

$$Q_{k,i} = Im\{\mathbf{S}_{k,i}\}$$

   - Create $f(x_i)$ from $\mathbf{P}_i$ and $\mathbf{Q}_i$ vectors
   - Calculate ***power mismatch***, $\Delta\mathbf{y}_i$

$$\Delta\mathbf{y}_i = \mathbf{y} - \mathbf{f}(\mathbf{x}_i)$$

   - Compute the Jacobian, $\mathbf{J}_i$, using voltage magnitudes and phase angles from $\mathbf{V}_i$

Solving the Power-Flow Problem - Procedure

6. while ($\varepsilon > reltol$) && ($i < max\_iter$) – cont'd
   - Solve for $\Delta\mathbf{x}_i$ using Gaussian elimination

$$\Delta\mathbf{y}_i = \mathbf{J}_i \Delta\mathbf{x}_i$$

     - Use the mldivide (\, backslash) operator in MATLAB: $\Delta\mathbf{x}_i = \mathbf{J}_i \backslash \Delta\mathbf{y}_i$
   - Update $\mathbf{x}$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta\mathbf{x}_i$$

   - Check for convergence using ***power mismatch***

$$\varepsilon_{i+1} = \max\left|\frac{y_k - f_k(\mathbf{x})}{y_k}\right|$$

   - Update the number of iterations

$$i = i + 1$$

The Jacobian Matrix

- The Jacobian matrix has four quadrants of varying dimension depending on the number of different types of buses:

$$
\begin{array}{cc}
 & \begin{array}{cc} \quad n_{PV}+n_{PQ} \quad & \quad n_{PQ} \quad \end{array} \\
\mathbf{J} = & \begin{bmatrix} \dfrac{\partial \mathbf{P}}{\partial \boldsymbol{\delta}} \; \boxed{\mathrm{J1}} & \dfrac{\partial \mathbf{P}}{\partial \mathbf{V}} \; \boxed{\mathrm{J2}} \\ \dfrac{\partial \mathbf{Q}}{\partial \boldsymbol{\delta}} \; \boxed{\mathrm{J3}} & \dfrac{\partial \mathbf{Q}}{\partial \mathbf{V}} \; \boxed{\mathrm{J4}} \end{bmatrix} \begin{array}{c} n_{PV}+n_{PQ} \\ \\ n_{PQ} \end{array}
\end{array}
$$

The Jacobian Matrix

- Jacobian elements are partial derivatives of (9) and (10) with respect to δ or V
- Formulas for the Jacobian elements:
  - $\underline{n \neq k}$

$$\mathbf{J}1_{kn} = \frac{\partial P_k}{\partial \delta_n} = V_k Y_{kn} V_n \sin(\delta_k - \delta_n - \theta_{kn}) \qquad (27)$$

$$\mathbf{J}2_{kn} = \frac{\partial P_k}{\partial V_n} = V_k Y_{kn} \cos(\delta_k - \delta_n - \theta_{kn}) \qquad (28)$$

$$\mathbf{J}3_{kn} = \frac{\partial Q_k}{\partial \delta_n} = -V_k Y_{kn} V_n \cos(\delta_k - \delta_n - \theta_{kn}) \qquad (29)$$

$$\mathbf{J}4_{kn} = \frac{\partial Q_k}{\partial V_n} = V_k Y_{kn} \sin(\delta_k - \delta_n - \theta_{kn}) \qquad (30)$$

The Jacobian Matrix

- Formulas for the Jacobian elements, cont'd:
  - $\underline{n = k}$

$$\mathbf{J}1_{kk} = \frac{\partial P_k}{\partial \delta_k} = -V_k \sum_{\substack{n=1 \\ n \neq k}}^{N} Y_{kn} V_n \sin(\delta_k - \delta_n - \theta_{kn}) \quad (31)$$

$$\mathbf{J}2_{kk} = \frac{\partial P_k}{\partial V_k} = V_k Y_{kk} \cos(\theta_{kk}) + \sum_{n=1}^{N} Y_{kn} V_n \cos(\delta_k - \delta_n - \theta_{kn}) \quad (32)$$

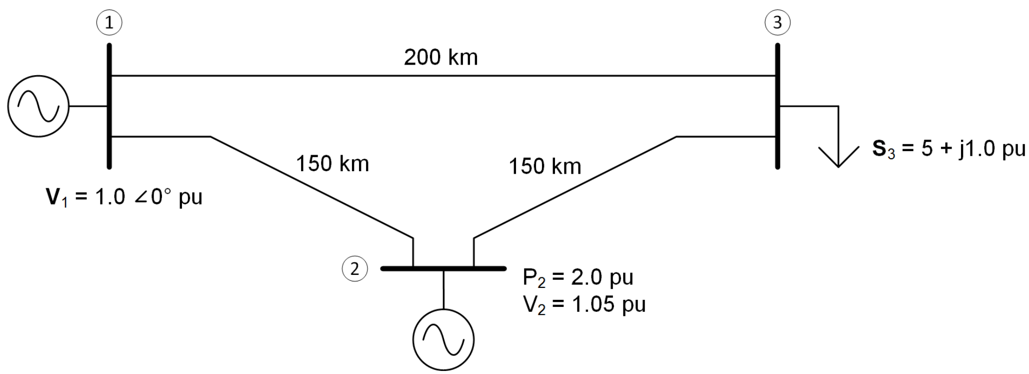
$$\mathbf{J}3_{kk} = \frac{\partial Q_k}{\partial \delta_k} = V_k \sum_{\substack{n=1 \\ n \neq k}}^{N} Y_{kn} V_n \cos(\delta_k - \delta_n - \theta_{kn}) \quad (33)$$

$$\mathbf{J}4_{kk} = \frac{\partial Q_k}{\partial V_k} = -V_k Y_{kk} \sin(\theta_{kk}) + \sum_{n=1}^{N} Y_{kn} V_n \sin(\delta_k - \delta_n - \theta_{kn}) \quad (34)$$

Power-Flow Solution – Example

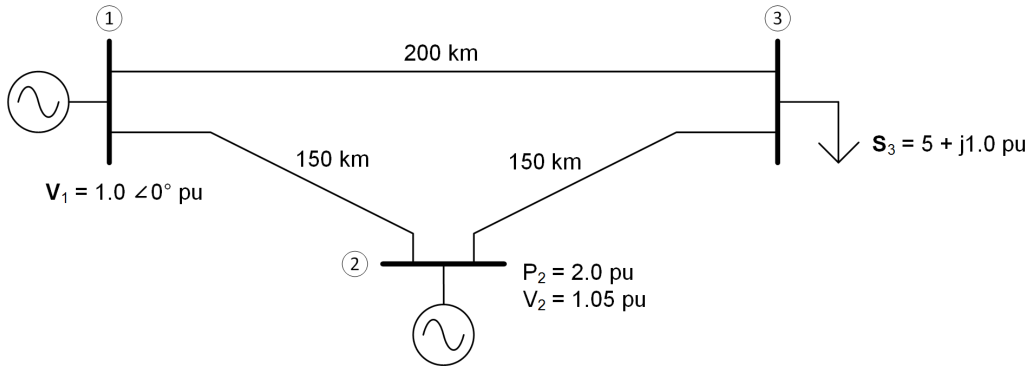
Power-Flow Solution – Buses

- Determine all bus voltages and power flows for the following three-bus power system



- Three buses, $n_{PV} = 1$, $n_{PQ} = 1$, ordered PV first, then PQ:
  - Bus 1: slack bus
    - V_1 and δ_1 are known, find P_1 and Q_1
  - Bus 2: PV bus
    - P_2 and V_2 are known, find δ_2 and Q_2
  - Bus 3: PQ bus
    - P_3 and Q_3 are known, find V_3 and δ_3

Power-Flow Solution – Admittance Matrix



- Per-unit, per-length impedance of all transmission lines:

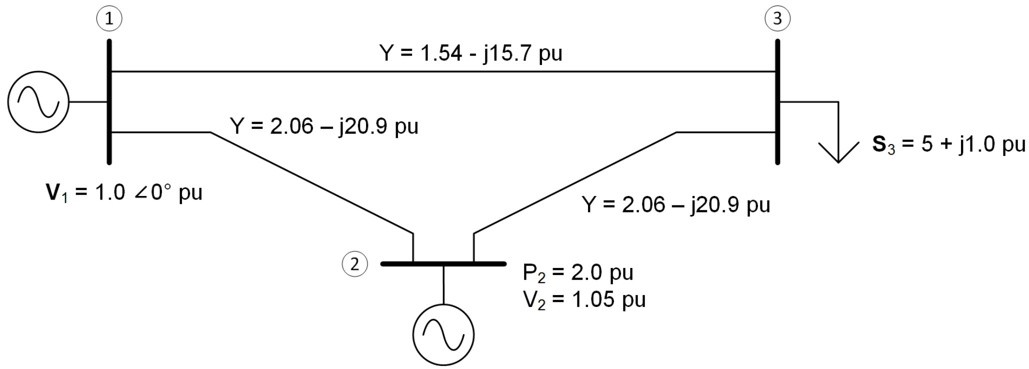
$$z = (31.1 + j316) \times 10^{-6}\ pu/km$$

  - Admittance of each line:

$$Y_{12} = Y_{23} = \frac{1}{z \cdot 150\ km} = 2.06 - j20.9\ pu$$

$$Y_{13} = \frac{1}{z \cdot 200\ km} = 1.54 - j15.7\ pu$$

Power-Flow Solution – Admittance Matrix



- The admittance matrix (see p. 8):

$$\mathbf{Y} = \begin{bmatrix} Y_{11} & -Y_{12} & -Y_{13} \\ -Y_{21} & Y_{22} & -Y_{23} \\ -Y_{31} & -Y_{32} & Y_{33} \end{bmatrix} = \begin{bmatrix} 3.6 - j36.6 & -2.06 + j20.9 & -1.5 + j15.7 \\ -2.06 + j20.9 & 4.1 - j41.8 & -2.06 + j20.9 \\ -1.5 + j15.7 & -2.06 + j20.9 & 3.6 - j36.6 \end{bmatrix}$$

- Admittance magnitude and angle matrices:

$$\mathrm{Y} = |\mathbf{Y}| = \begin{bmatrix} 36.8 & 21.0 & 15.8 \\ 21.0 & 42.0 & 21.0 \\ 15.8 & 21.0 & 36.8 \end{bmatrix}, \qquad \boldsymbol{\theta} = \begin{bmatrix} -84.4° & 95.6° & 95.6° \\ 95.6° & -84.4° & 95.6° \\ 95.6° & 95.6° & -84.4° \end{bmatrix}$$

Power-Flow Solution – Initialize Knowns

- Known quantities
  - Slack bus: $V_1 = 1.0\, pu$, $\delta_1 = 0°$
  - PV bus: $V_2 = 1.05\, pu$, $P_2 = 2.0\, pu$
  - PQ bus: $P_3 = -5.0\, pu$, $Q_3 = -1.0\, pu$
  - Output vector

$$\mathbf{y} = \begin{bmatrix} \mathbf{P} \\ \mathbf{Q} \end{bmatrix} = \begin{bmatrix} P_2 \\ P_3 \\ Q_3 \end{bmatrix} = \begin{bmatrix} 2.0 \\ -5.0 \\ -1.0 \end{bmatrix}$$

Power-Flow Solution – Initialize Unknowns

- The vector of unknown quantities to be solved for is

$$\mathbf{x} = \begin{bmatrix} \boldsymbol{\delta} \\ \mathbf{V} \end{bmatrix} = \begin{bmatrix} \delta_2 \\ \delta_3 \\ V_3 \end{bmatrix}$$

- Initialize all unknown bus voltage phasors to $\mathbf{V}_k = 1.0\angle 0°\ pu$

$$\mathbf{x}_0 = \begin{bmatrix} \boldsymbol{\delta}_0 \\ \mathbf{V}_0 \end{bmatrix} = \begin{bmatrix} \delta_{2,0} \\ \delta_{3,0} \\ V_{3,0} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1.0 \end{bmatrix}$$

- The complete vector of bus voltage phasors – partly known, partly unknown – is

$$\mathbf{V} = \begin{bmatrix} V_1\angle\delta_1 \\ V_2\angle\delta_2 \\ V_3\angle\delta_3 \end{bmatrix} = \begin{bmatrix} 1.0\angle 0° \\ 1.05\angle\delta_{2,0} \\ V_{3,0}\angle\delta_{3,0} \end{bmatrix} = \begin{bmatrix} 1.0\angle 0° \\ 1.05\angle 0° \\ 1.0\angle 0° \end{bmatrix}$$

Power-Flow Solution – Jacobian Matrix

- The Jacobian matrix for this system is

$$\mathbf{J} = \begin{bmatrix} \frac{\partial P_2}{\partial \delta_2} & \frac{\partial P_2}{\partial \delta_3} & \frac{\partial P_2}{\partial V_3} \\ \frac{\partial P_3}{\partial \delta_2} & \frac{\partial P_3}{\partial \delta_3} & \frac{\partial P_3}{\partial V_3} \\ \frac{\partial Q_3}{\partial \delta_2} & \frac{\partial Q_3}{\partial \delta_3} & \frac{\partial Q_3}{\partial V_3} \end{bmatrix}$$

- This matrix will be computed on each iteration using the current approximation to the vector of unknowns, $\mathbf{x}_i$

Power-Flow Solution – Set Up and Iterate

- Set up N-R iteration parameters
  - *reltol* = 1e-6
  - *max_iter* = 1e3
  - ε_0 = 10
  - i = 0
- Iteratively update the approximation to the vector of unknowns as long as
  - Stopping criterion is not satisfied

$$\varepsilon_i > \varepsilon_s$$

  - Maximum number of iterations is not exceeded

$$i \leq max\_iter$$

Power-Flow Solution – Iterate

- $\underline{i = 0}$:
  - Vector of bus voltage phasors

$$\mathbf{V}_0 = \begin{bmatrix} V_1\angle\delta_1 \\ V_2\angle\delta_{2,0} \\ V_{3,0}\angle\delta_{3,0} \end{bmatrix} = \begin{bmatrix} 1.0\angle0° \\ 1.05\angle0° \\ 1.0\angle0° \end{bmatrix}$$

  - Current injected into each bus

$$\mathbf{I}_0 = \mathbf{Y} \cdot \mathbf{V}_0$$

$$\mathbf{I}_0 = \begin{bmatrix} 3.6 - j36.6 & -2.1 + j20.9 & -1.5 + j15.7 \\ -2.1 + j20.9 & 4.1 - j41.8 & -2.1 + j20.9 \\ -1.5 + j15.7 & -2.1 + j20.9 & 3.6 - j36.6 \end{bmatrix} \begin{bmatrix} 1.0\angle0° \\ 1.05\angle0° \\ 1.0\angle0° \end{bmatrix}$$

$$\mathbf{I}_0 = \begin{bmatrix} 1.05\angle95.6° \\ 2.10\angle-84.4° \\ 1.05\angle95.6° \end{bmatrix}$$

Power-Flow Solution – Iterate

- $\underline{i = 0}$:
  - Complex power injected into each bus

$$\mathbf{S}_0 = \mathbf{V}_0 \;.\!* \;\mathbf{I}_0^*$$

$$\mathbf{S}_0 = \begin{bmatrix} 1.0\angle 0° \\ 1.05\angle 0° \\ 1.0\angle 0° \end{bmatrix} .* \begin{bmatrix} 1.05\angle 95.6° \\ 2.10\angle -84.4° \\ 1.05\angle 95.6° \end{bmatrix}^*$$

$$\mathbf{S}_0 = \begin{bmatrix} -0.103 - j1.045 \\ 0.216 + j2.195 \\ -0.103 - j1.045 \end{bmatrix}$$

  - Real and reactive power

$$\mathbf{P}_0 = \begin{bmatrix} -0.103 \\ 0.216 \\ -0.103 \end{bmatrix}, \qquad \mathbf{Q}_0 = \begin{bmatrix} -1.045 \\ 2.195 \\ -1.045 \end{bmatrix}$$

Power-Flow Solution – Iterate

- $\underline{i = 0}$:
  - Power mismatch

$$\Delta\mathbf{y}_0 = \mathbf{y} - \mathbf{f}(\mathbf{x}_0)$$

$$\Delta\mathbf{y}_0 = \begin{bmatrix} 2.0 \\ -5.0 \\ -1.0 \end{bmatrix} - \begin{bmatrix} 0.216 \\ -0.103 \\ -1.045 \end{bmatrix} = \begin{bmatrix} 1.784 \\ -4.897 \\ 0.045 \end{bmatrix}$$

  - Next, compute the Jacobian matrix

$$\mathbf{J}_0 = \begin{bmatrix} \frac{\partial P_2}{\partial \delta_2} & \frac{\partial P_2}{\partial \delta_3} & \frac{\partial P_2}{\partial V_3} \\ \frac{\partial P_3}{\partial \delta_2} & \frac{\partial P_3}{\partial \delta_3} & \frac{\partial P_3}{\partial V_3} \\ \frac{\partial Q_3}{\partial \delta_2} & \frac{\partial Q_3}{\partial \delta_3} & \frac{\partial Q_3}{\partial V_3} \end{bmatrix}_{\mathbf{x}=\mathbf{x}_0}$$

Power-Flow Solution – Iterate

- $\underline{i = 0}$:
  - Elements of the Jacobian matrix are computed using V and δ values from $\mathbf{V}_0$ and Y and θ values from $\mathbf{Y}$:

$$V_0 = \begin{bmatrix} 1.0 \\ 1.05 \\ 1.0 \end{bmatrix}$$

$$\delta_0 = \begin{bmatrix} 0° \\ 0° \\ 0° \end{bmatrix}$$

$$Y = \begin{bmatrix} 36.8 & 21.0 & 15.8 \\ 21.0 & 42.0 & 21.0 \\ 15.8 & 21.0 & 36.8 \end{bmatrix}$$

$$\theta = \begin{bmatrix} -84.4° & 95.6° & 95.6° \\ 95.6° & -84.4° & 95.6° \\ 95.6° & 95.6° & -84.4° \end{bmatrix}$$

Power-Flow Solution – Iterate

- $\underline{i = 0}$:
  - Jacobian, $\mathbf{J}1$

$$\frac{\partial P_2}{\partial \delta_2} = -V_2\left(Y_{21}V_1 \sin(\delta_2 - \delta_1 - \theta_{21}) + Y_{23}V_3 \sin(\delta_2 - \delta_3 - \theta_{23})\right)$$

$$\frac{\partial P_3}{\partial \delta_3} = -V_3\left(Y_{31}V_1 \sin(\delta_3 - \delta_1 - \theta_{31}) + Y_{32}V_2 \sin(\delta_3 - \delta_2 - \theta_{32})\right)$$

$$\frac{\partial P_2}{\partial \delta_3} = V_2 Y_{23} V_3 \sin(\delta_2 - \delta_3 - \theta_{23})$$

$$\frac{\partial P_3}{\partial \delta_2} = V_3 Y_{32} V_2 \sin(\delta_3 - \delta_2 - \theta_{32})$$

Power-Flow Solution – Iterate

- $\underline{i = 0}$:
  - Jacobian, $\mathbf{J}2$

$$\frac{\partial P_2}{\partial V_3} = V_2 Y_{23} \cos(\delta_2 - \delta_3 - \theta_{23})$$

$$\frac{\partial P_3}{\partial V_3} = 2 \cdot V_3 Y_{33} \cos(\theta_{33}) +$$

$$Y_{31} V_1 \cos(\delta_3 - \delta_1 - \theta_{31}) + Y_{32} V_2 \cos(\delta_3 - \delta_2 - \theta_{32})$$

  - Jacobian, $\mathbf{J}3$

$$\frac{\partial Q_3}{\partial \delta_2} = -V_3 Y_{32} V_2 \cos(\delta_3 - \delta_2 - \theta_{32})$$

$$\frac{\partial Q_3}{\partial \delta_3} = V_3 (Y_{31} V_1 \cos(\delta_3 - \delta_1 - \theta_{31}) + Y_{32} V_2 \cos(\delta_3 - \delta_2 - \theta_{32}))$$

Power-Flow Solution – Iterate

- $\underline{i = 0}$:
  - Jacobian, $\mathbf{J}4$

$$\frac{\partial Q_3}{\partial V_3} = V_3 Y_{33}\cos(\theta_{33}) +$$

$$Y_{31} V_1 \cos(\delta_3 - \delta_1 - \theta_{31}) + Y_{32} V_2 \cos(\delta_3 - \delta_2 - \theta_{32})$$

  - Evaluating the Jacobian expressions using V and δ values from $\mathbf{V}_0$ and Y and θ values from $\mathbf{Y}$, gives

$$\mathbf{J}_0 = \begin{bmatrix} 43.89 & -21.95 & -2.160 \\ -21.95 & 37.62 & 3.497 \\ 2.160 & -3.702 & 35.53 \end{bmatrix}$$

Power-Flow Solution – Iterate

- $\underline{i = 0}$:
  - Use Gaussian elimination to solve for $\Delta\mathbf{x}_0$

$$\Delta\mathbf{y}_0 = \mathbf{J}_0\Delta\mathbf{x}_0 = \begin{bmatrix} 43.89 & -21.95 & -2.160 \\ -21.95 & 37.62 & 3.497 \\ 2.160 & -3.702 & 35.53 \end{bmatrix} \begin{bmatrix} \Delta x_{1,0} \\ \Delta x_{2,0} \\ \Delta x_{3,0} \end{bmatrix} = \begin{bmatrix} 1.784 \\ -4.897 \\ 0.045 \end{bmatrix}$$

$$\Delta\mathbf{x}_0 = \begin{bmatrix} -0.0345 \\ -0.1492 \\ -0.0122 \end{bmatrix}$$

  - Update the vector of unknowns, $\mathbf{x}$

$$\mathbf{x}_1 = \mathbf{x}_0 + \Delta\mathbf{x}_0 = \begin{bmatrix} 0 \\ 0 \\ 1.0 \end{bmatrix} + \begin{bmatrix} -0.0345 \\ -0.1492 \\ -0.0122 \end{bmatrix} = \begin{bmatrix} -0.0345 \\ -0.1492 \\ 0.9878 \end{bmatrix}$$

Power-Flow Solution – Iterate

- $\underline{i = 0}$:
  - Use power mismatch to check for convergence

$$\varepsilon_0 = \max\left|\frac{y_k - f_k(x)}{y_k}\right| = 0.9794$$

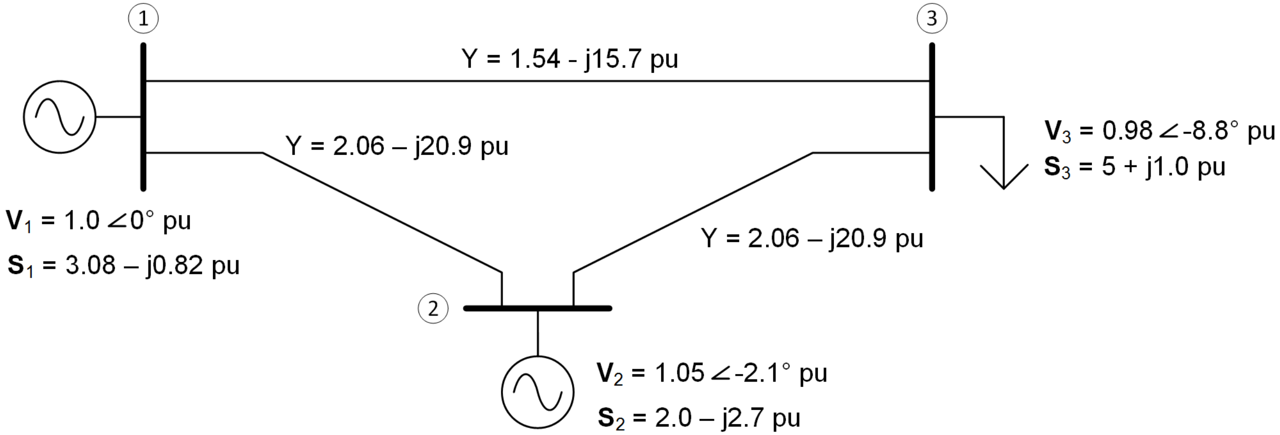
  - Move on to the next iteration, $i = 1$
    - Create $\mathbf{V}_1$ using $\mathbf{x}_1$ values
    - Calculate $\mathbf{I}_1$
    - Calculate $\mathbf{S}_1$, $\mathbf{P}_1$, $\mathbf{Q}_1$
    - Create $\mathbf{f}(\mathbf{x}_1)$ from $\mathbf{P}_1$ and $\mathbf{Q}_1$
    - Calculate $\Delta\mathbf{y}_1$, $\mathbf{J}_1$, $\Delta\mathbf{x}_1$
    - Update $\mathbf{x}$ to $\mathbf{x}_2$
    - Check for convergence
    - …

Power-Flow Solution

- Convergence is achieved after four iterations

$$\mathbf{V}_4 = \begin{bmatrix} 1.0\angle 0° \\ 1.1\angle -2.1° \\ 0.97\angle -8.8° \end{bmatrix}, \qquad \mathbf{S}_4 = \begin{bmatrix} 3.08 - j0.82 \\ 2.0 + j2.67 \\ -5.0 - j1.0 \end{bmatrix}$$

$$\varepsilon_4 = 0.41 \times 10^{-6}$$



Example Problems

For the power system shown, determine

a) The type of each bus
b) The first row of the admittance matrix, $\mathbf{Y}$
c) The vector of unknowns, $\mathbf{x}$
d) The vector of knowns, $\mathbf{y}$
e) The Jacobian matrix, $\mathbf{J}$, in symbolic form